

(19)

Europäisches Patentamt  
European Patent Office  
Office européen des brevets



(11)

EP 1 221 653 A2

(12)

## EUROPEAN PATENT APPLICATION

(43) Date of publication:  
10.07.2002 Bulletin 2002/28

(51) Int Cl. 7: G06F 9/445

(21) Application number: 01130482.1

(22) Date of filing: 20.12.2001

(84) Designated Contracting States:  
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU  
MC NL PT SE TR  
Designated Extension States:  
AL LT LV MK RO SI

(30) Priority: 21.12.2000 US 745404

(71) Applicant: MICROSOFT CORPORATION  
Redmond, Washington 98052-6399 (US)

(72) Inventors:  
• Ray, Kenneth D.  
Redmond, Washington 98052 (US)  
• Bhesania, Firdosh K.  
Kirkland, Washington 98033 (US)  
• Dunn, John C.  
Issaquah, Washington 98033 (US)

(74) Representative: Grünecker, Kinkeldey,  
Stockmair & Schwanhäußer Anwaltssozietät  
Maximilianstrasse 58  
80538 München (DE)

(54) System and method to specify extended configuration descriptor information in USB devices

(57) The following system and procedure for specifying an extended configuration descriptor includes a USB device that responds to device requests from a host. In response to receiving a host-specific device request that specifies a predetermined index, the USB device returns an extended configuration descriptor to the

requestor. The extended configuration descriptor includes information that can be used by the requestor to control the USB device. When the USB device is a composite device, the extended configuration descriptor includes function information corresponding to a plurality of sub-devices. Each function comprises one or more interfaces.

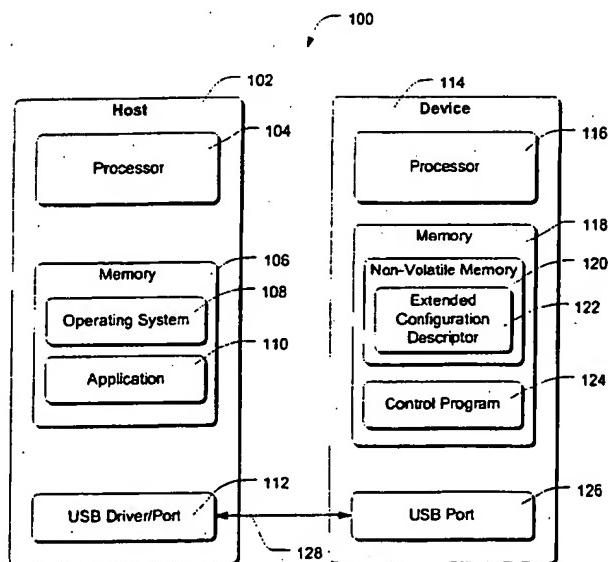


Fig. 1

EP 1 221 653 A2

**Description****RELATED APPLICATIONS**

- 5 [0001] This application is related to a prior US Patent Application filed February 4, 2000, titled "Host-Specified USB Device Requests", serial number 09/498,056, which is hereby incorporated by reference.

**TECHNICAL FIELD**

- 10 [0002] The following description relates generally to the use of peripheral devices with software applications. More specifically, the following description relates to the use of device specific information and resources with such software applications.

**BACKGROUND OF THE INVENTION**

15 [0003] The Universal Serial Bus (USB) is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripheral devices. The attached peripheral devices share USB bandwidth through a host-scheduled, token-based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals are in operation.

20 [0004] The USB is defined by a specification that is approved by a committee of industry representatives. The specification covers all aspects of USB operation, including electrical, mechanical, and communications characteristics. To be called a USB device, a peripheral must conform to this very exacting specification.

[0005] USB device information is typically stored in so-called "descriptors" or request codes—data structures formatted as specified by the USB specification. Descriptors are used in a USB system to define "device requests" from a host to a peripheral device. A device request is a data structure that is conveyed in a "control transfer" from the host to the peripheral device. A control transfer contains the following fields:

- *bmRequestType*—a mask field indicating (a) the direction of data transfer in a subsequent phase of the control transfer; (b) a request type (standard, class, vendor, or reserved); and (c) a recipient (device, interface, endpoint, or other). The primary types of requests specified in the "request type" field are the "standard" and "vendor" types, which will be discussed below.
- *bRequest*—a request code indicating one of a plurality of different commands to which the device is responsive.
- *wValue*—a field that varies according to the request specified by *bRequest*.
- *wIndex*—a field that varies according to request; typically used to pass an index or offset as part of the specified request.
- *wLength*—number of bytes to transfer if there is a subsequent data stage.

[0006] All USB devices are supposed to support and respond to "standard" requests—referred to herein as "USB-specific" requests. In USB-specific requests, the request type portion of the *bmRequestType* field contains a predefined value indicative of the "standard" request type.

[0007] Each different USB-specific request has a pre-assigned USB-specific request code, defined in the USB specification. This is the value used in the *bRequest* field of the device request, to differentiate between different USB-specific requests. For each USB-specific request code, the USB specification sets forth the meanings of *wValue* and *wIndex*, as well as the format of any returned data.

[0008] USB devices can optionally support "vendor" requests—referred to herein as "device-specific" requests. In a device-specific request, the request type portion of the *bmRequestType* field contains a predefined value to indicate a "vendor" request type. In the case of device-specific requests, the USB specification does not assign request codes, define the meanings of *wValue* and *wIndex*, or define the format of returned data. Rather, each device has nearly complete control over the meaning, functionality, and data format of device-specific requests. Specifically, the device can define its own requests and assign device-specified request codes to them. This allows devices to implement their own device requests for use by host computers, and provides tremendous flexibility for manufacturers of peripherals.

[0009] The inventors have discovered a need for a similar feature that would benefit various hosts, application programs, host operating systems hardware manufacturers (OEMs), and Independent Hardware Vendors (IHVs). Specifically, designers of application programs and operating systems would value the opportunity to define their own device requests (and the associated responses), and to have such requests supported in a uniform way by compatible peripherals. Moreover, OEMs and IHVs would value the ability to supply additional USB device specific information to the hosts, application programs and host operating systems in response to such device requests. However, the different request types supported in the *bmRequestType* field of a USB device request do not include a "host" type of request.

- [0010] As an example of this need for a host type of request, consider that the USB Device Working Group (DWG) has defined a set of standards that describe different types of USB devices. For each of these standards, the DWG has created a respective class specification that specifies static predetermined class and subclass numbers that correspond to the class specification. These specifications are designed such that to be in compliance with a particular class specification, an operating system must include only a single default, or "generic" device driver to control a USB device that is configured according to the particular class specification.
- [0011] When a USB device is installed onto a system, the device indicates that it was created according to a particular DWG class specification by listing predefined class and subclass codes that correspond to the DWG specification. These predefined codes can be used by an operating system to load a single generic device driver to control the device.
- [0012] However, it is often the case that a vendor of a USB device would prefer that a different, or "more specific" device driver be used to control the device, rather than the generic device driver supplied by the operating system. For example, a more specific device driver may be preferred to work around known bugs in the device's software, provide product identification, use value added features, and the like. Such more specific device drivers are typically supplied by OEM/IHV supplied installation media that are distributed with a device. Such installation media include installation disks (e.g., floppy diskettes) or setup computer program applications.
- [0013] To provide for more specific device driver matches with respect to a USB device, USB device descriptors include markings that describe the manufacturer, product, and revision. An operating system (OS) can use such markings to determine whether a more specific, and therefore a more customized device driver match is available from the installation media. If for some reason a specific match is not found, the concepts of class and subclass codes, as described above, facilitate the loading of a generic device driver to control the device.
- [0014] One of the benefits of using a USB device is the reduced amount of interaction typically required of a user to attach and configure a device. This has been a strong point of USB and has lead to increasing sales of USB devices over the past few years. Thus, being able to load a device driver without having to use additional OEM/IHV supplied installation media to configure the device greatly simplifies end user USB device attachment scenarios.
- [0015] Unfortunately, unless the USB DWG has already allocated certain class and subclass codes for a particular USB device, that particular device cannot have a match based on class and subclass codes to identify a generic device driver to control the device. If the device's corresponding class and subclass codes are not supported by the DWG, an OEM/IHV distributing the device must at least provide installation media with the device to specify a given match that corresponds to a generic driver that is already present on the system. Otherwise, the OEM/IHV must supply a special device driver with the corresponding installation media.
- [0016] In light of the above, a user of a device that specifies non-standard DWG class codes is required to use such installation media to load a device driver to control the device. It would be beneficial both in terms of simplifying matters for OEMs/IHVs and simplifying customer ease of use if an operating system could specify a default device driver that is already present on the system in response to installing a device that does not specify a standard USB DWG class and/or subclass code.
- [0017] Ideally, an operating system vendor could develop a new standard USB device class—one not supported by DWG, and distribute a device driver to control devices built to that new standard with the operating system. In this manner, an OEM/IHV could not only distribute devices built to that standard before the USB DWG has adopted a corresponding class and subclass code, but also distribute such devices without needing to supply a special device driver and corresponding installation media. In this way the operating system would natively support the new non-standard USB device by providing a generic device driver to control the new device.
- [0018] To natively support a non-standard USB device, an operating system would have to request specific information from the new device to identify the new device's non-standard class and subclass codes to determine the particular generic device driver to control the new device. However, in response to device specific requests from an operating system, a USB device (as currently defined by the USB specifications) can only return information related to vendor identification, product identification, and revision information. There is no standard request to allow a vendor to return additional information to an operating system to indicate that the device supports non-standard class and/or subclass codes—those not supported by the USB DWG.
- [0019] Thus, it would be beneficial if a USB device could indicate through a predefined USB request that it supports such non-standard class and/or subclass codes. This information could be used by an operating system to install default device drivers to control the USB device. However, as discussed above, such a USB request is not defined by the USB specification.
- [0020] In yet another example that illustrates the need for a host specific USB request, consider that the DWG has established a number of well-defined requests to obtain interfaces that correspond to functions in certain composite devices. A composite device includes a number of logical devices (LDs), wherein each LD is a fully functional sub-device of the composite device. There is typically a one-to-one correspondence between a function and an interface in a composite device, with the exception of certain devices such as USB speaker devices, which implement more than one interface for a single function. In such exceptional cases, because the DWG has a predefined request to

return the multiple interfaces that correspond to a function, each interface can be returned in response to a single request.

- [0021] Otherwise, when the DWG has not defined such a request, a complex procedure must typically be implemented to determine each of the interfaces that correspond to respective functions in a composite device. For example, to determine if a USB device is a composite device, a host must first request the device to indicate how many functions, or LDs that it supports. In response, a composite device will indicate its ability to support multiple functions by first identifying the number of logical devices (LDs) that the device supports. Next, to obtain specific information corresponding to each supported LD, the host must typically issue separate requests, each request being overloaded with a respective LD identification, to obtain each respective LD descriptor. Thus, a typical composite USB device would not return information corresponding to each interface grouping defined in the composite device in response to a single query unless the single query was already predefined by the DWG to return more than one interface in response to a single request.

[0022] It would be beneficial to USB composite device makers and to software developers if there was a simple way for a USB device to specify the interfaces that comprise a single function, such that the interfaces could be determined without the need to implement the complex procedure discussed above. In this manner, newer composite devices not yet supported by the DWG could be treated exactly as the ones currently supported by the DWG.

[0023] Accordingly, the invention arose out of concerns associated with providing a host-specific device request that solves the problems described above.

## **20 SUMMARY**

- [0024] One embodiment of the described system and procedure specifies non-standard compatible IDs, or non-standard class and subclass codes in a USB device in response to a host-specific device request. A non-standard compatible ID is a class or subclass code that is not supported or defined by the USB DWG. The USB device comprises an extended configuration descriptor in firmware of the USB device. The extended configuration descriptor comprises a set of non-standard class codes that can be used by a host operating system to identify one or more device drivers to control the device. Responsive to receiving a host-specific device request from the operating system, the USB device communicates the extended configuration descriptor to a requestor. Thus, the system and procedure of this implementation simplifies end-user device installation scenarios and allows OEMs/IHVs to implement new USB devices without distributing special device drivers and/or corresponding installation media with each single new USB device.
- [0025] Moreover, the extended configuration descriptor includes a header section and one or more control function sections. The header section indicates the number of control functions for which mappings exist in the extended configuration descriptor. Each control function section indicates information corresponding to a respective function for the USB device. Each control function comprises one or more interfaces. Because the extended configuration descriptor is communicated to a requestor responsive to the receipt of a single host-specific device request, the inventive concepts of this implementation provide a simple way for a new USB device to specify a group of interfaces that comprise a single function.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

### **40 [0026]**

- Fig. 1 is a block diagram of an exemplary host/peripheral USB system.  
 Fig. 2 is a flowchart diagram that shows top-level methodological aspects of an exemplary procedure to obtain and use a device-specific request code for host-specific device requests.  
 Fig. 3 is a flowchart diagram that shows detailed aspects of an exemplary procedure to obtain a device-specific request code.  
 Fig. 4 is a block diagram showing aspects of an exemplary data structure for an extended configuration descriptor.

## **50 DETAILED DESCRIPTION**

- [0027] The following description sets forth a specific embodiment of a system and procedure that incorporates elements recited in the appended claims. The embodiment is described with specificity in order to meet statutory requirements. However, the description itself is not intended to limit the scope of this patent. Rather, the inventors have contemplated that the claimed subject matter might also be embodied in other ways, to include different elements or combinations of elements similar to the ones described in this document, in conjunction with other present or future technologies.

Exemplary System

- [0028] Fig. 1 shows a system 100 wherein a USB device includes non-standard compatible IDs, or non-standard class and subclass codes. Additionally, the system includes a host application program and host operating system that is able to enumerate non-standard compatible IDs, or non-standard class and subclass code corresponding to USB devices. System 100 is compatible with the Universal Serial Bus (USB) specifications. These specifications are available from USB Implementers Forum, which has current administrative headquarters in Portland, Oregon (current Internet URL: [www.usb.org](http://www.usb.org)).  
 5
- [0029] System 100 includes a host computer 102 and a USB peripheral device 114. The respective functionality of the computer and peripheral device is embodied in many cases by computer-executable instructions, such as program modules, that are executed by respective processors. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types.  
 10
- [0030] Computer 102 is a conventional desktop PC or other type of computer. Computer 102 has one or more processors 104 and one or more forms of computer-readable memory media 106 such as electronic memory, magnetic storage media, optical storage media, or some other type of data storage. Programs are stored in memory 106 from where they are executed by processor 104. In this example, such programs include an operating system 108 such as the Microsoft "Windows® family of operating systems. The operating system provides various system services to one or more application programs 110 running on the computer.  
 15
- [0031] The computer also has a USB communications driver and port 112. The USB port is supported by operating system 108. To communicate with a USB device, an application program 110 makes high-level calls to system services provided by the operating system. The system services take care of lower level communications details, and return requested information to the application program.  
 20
- [0032] Peripheral device 114 is one of any number of different types of USB devices such as a data storage device, a digital camera, a scanner, a joystick, game pad, steering unit, mouse, stylus, digital speaker, microphone, display device, and the like. Peripheral device 114 has one or more processors 116 and one or more forms of computer-readable memory media 118, including at least some form of non-volatile memory media 120.  
 25
- [0033] The peripheral device 114 has a USB port 126 and communicates with computer 102 via a USB communications medium 128. The peripheral device also has operating logic 124, which is executed by processor 116 to detect control actuation and for communicating with computer 102 across communication path 128.  
 30
- [0034] The peripheral device 114 responds to requests from the host computer 102 across the communication path 128. These requests are made using control transfers where setup packets (not shown) are exchanged. The USB device returns descriptors in response to exchanging such setup packets. Although the USB Specification defines a number of different standard class and vendor specific descriptors, an extended configuration descriptor 122 is not defined in the USB specification. The extended configuration descriptor allows OEMs/IHVs to store non-standard class and subclass codes—those that are not yet defined by the USB DWG, into non-volatile memory 120 of the device. Moreover, the extended configuration descriptor provides a simple way for a composite USB device to specify a group of interfaces that comprise a single function.  
 35
- [0035] Extended configuration descriptor 122 provides non-standard class codes and subclass codes on a per function basis. (Device vendors can build multiple functions into a device). A function is an independently controlled aspect of a device. A function can be considered to be a group of interfaces conglomerated together that serve a similar purpose. Class codes are also referred to herein as "compatible ids". Subclass codes are also referred to herein as "sub-compatible ids".  
 40
- [0036] Peripheral device 114 supports host-specific device requests to obtain information that is not specified in the USB specification. In this implementation, prior to using the peripheral device, the computer sends a host-specific request to the peripheral device requesting class codes and/or subclass codes that correspond to the peripheral device. In response to the request, the peripheral device provides extended configuration descriptor 122 to the computer. Within the extended configuration descriptor, the computer locates data corresponding to the peripheral device 114, and extracts non-standard class and/or subclass codes, as well as one or more interfaces that correspond to one or more functions. The operating system then loads device drivers based on the extracted extended configuration descriptor to control and interface with the peripheral device.  
 45  
 50

Exemplary Procedure to Obtain a Device Specific Request Code

- [0037] Fig. 2 shows top-level methodological aspects of an exemplary procedure to obtain a device-specific request code for a computer to use when making a host-specific device request. Generally, a new non-standard USB-specific device request is defined for use with various USB peripherals. This request is referred to herein as a host-specific device request. Because of the described methodology, the host-specific device request can be defined by the manufacturer of an operating system or application program, and can then be made available to peripheral vendors so that  
 55

they can support and respond to the newly defined request. As an example, an OS manufacturer might define a new descriptor allowing peripherals to return device specific data and resources to an operating system in a data format that is determined by the operating system. This would allow the operating system to use a single device request to obtain this information from various different peripherals (assuming those peripherals are configured to support the new device request).

[0038] In an initialization phase 200, the host sends a request to the peripheral in the form of a USB-specified device request. The request is for a device-specific request code—of the device's choosing—that will be subsequently used as the request code for the host-specific device request.

[0039] Once this request code is obtained, it is used in a subsequent phase 202 to initiate the host-specified device request to obtain the extended configuration descriptor 122 (see Fig. 1). Specifically, the host specifies the request code as the *bRequest* value in a control transfer (see the "Background" section for details regarding a control transfer). The actual protocol of this device request (meanings of *bIndex*, *bValue*, etc.) is as specified in the definition of the host-specific device request. Phase 202 is repeated as desired during subsequent operation, without repetition of initialization phase 100.

[0040] Fig. 3 shows more details regarding the initialization phase 200. The host performs an action 300 of sending a GET\_DESCRIPTOR device request to the peripheral device. The GET\_DESCRIPTOR device request is a standard, USB-specific request, identified in a control transfer by the GET\_DESCRIPTOR request code (*bRequest* = GET\_DESCRIPTOR). The fields of the control transfer (discussed above in the background section) have values as follows:

- *bmRequestType* = 10000000 (binary), indicating a "device-to-host" transfer, a "standard" or "USB-specific" type request, and a device recipient.
- *bRequest* = GET\_DESCRIPTOR This is a constant (six) defined by the USB specification
- *wValue* = 03EE (hex). The high byte (03) indicates that the request is for a "string" descriptor, and the low byte is an index value that is predefined as a constant in the definition of the host-specified device request. In this example, it has been defined as EE (hex), but could be predefined as any other value.
- *wIndex* = 0.
- *wLength* = 12 (hex). This is the length of a host-specific request descriptor that will be returned in response to this request. In the described example, the length is 12 (hex).
- *data*—returned host-specific request descriptor.

[0041] A compatible USB device is configured to respond to a request such as this (where *wValue* = 03EE (hex)) by returning a host-specific request descriptor such as the extended configuration descriptor 122 of Fig. 1. This descriptor is not defined by the USB standard, but has fields as defined in the following discussion. The host-specific request descriptor designates a device-specific request code that will work on *this device* to initiate the host-specific request code. In other words, the manufacturer of the device can select any device-specific request code, and associate it with an implementation of the host-specific device request.

[0042] More specifically, the device receives the GET\_DESCRIPTOR device request (block 302) and performs a decision 304 regarding whether the index value (the second byte of *wValue*) corresponds to the predetermined value (EE (hex)). This predetermined value is a value that is chosen to be used specifically for this purpose.

[0043] If the index value does not correspond to the predetermined value, the device responds at 305 in an appropriate way, usually by returning some other descriptor that corresponds to the index value. If the index value does correspond to the predetermined value, an action 306 is performed of returning the host-specific request descriptor to the host.

[0044] The host-specific request descriptor includes, for example, the following fields:

- *bLength*—the length of the descriptor (12 (hex) in this example).
- *bDescriptorType*—the type of descriptor (string type in this example).
- *qwSignature*—a signature confirming that this descriptor is indeed a descriptor of the type requested. The signature optionally incorporates a version number. For example, in the described example MSFT100 indicates that this descriptor is for an "MSFT" host-specific device request, version "100" or 1.00.
- *bVendorCode*—the device-specific request code that is to be associated with the host-specified device request.
- *bPad*—a pad field of one byte.

[0045] At 308, the host receives the host-specific request descriptor and then performs an action 310 of checking or verifying the signature and version number found in the *qwSignature* field. The correct signature confirms that the device is configured to support host-specific request codes. If either the signature or version number is incorrect, at 311 the host assumes that the device does not support host-specific request codes, and no further attempts are made

to use this feature.

[0046] The signature field of the host-specific request descriptor block is what provides backward compatibility. A non-compatible device (one that doesn't support host-specific request codes) might use the predetermined *wValue* 03EE (hex) to store some other string descriptor, which will be returned to the host without any indication of problems.

5 However, this will become apparent to the host after it examines the data in the location where the signature is supposed to be. If the signature is not found, the host knows that the returned descriptor is not of the type requested, and will assume that the device does not support host-specific request codes.

[0047] If the signature and version are confirmed in block 310, at block 312 the host reads the device-specific request code from the *bVendorCode* field, and uses it in the future as a host-specific request code, to initiate the host-specific 10 device request. When using the device, the host sends the host-specific device request by specifying the obtained device-specific request code as part of a control transfer. The device responds by performing one or more predefined actions or functions that correspond to the host-specific device request, in accordance with the specification of the host-specific device request.

[0048] The host-specific device request itself is in the format of a normal USB control transfer, including the fields 15 discussed in the "Background" section above. The *bRequest* field is set to the value of the *bVendorCode* field of the host-specific request descriptor, which was earlier obtained from the peripheral device. The *bmRequestType* field is set to 11000001 (binary), indicating a device-to-host data transfer, a "vendor" or device-specific request type, and a device recipient.

[0049] The *wValue* and *wIndex* fields are used as defined by the definition of the host-specific device request. The 20 *wLength* field indicates the number of bytes to transfer if there is a subsequent data transfer phase in the device request.

[0050] In a presently preferred implementation of this system, the host-specific device request is used to request one of a plurality of available host-defined string descriptors from the device. The *wIndex* field of the host-specific device request indicates which of the plurality of strings are to be returned. The device returns the descriptor referred to by *wIndex*.

25 [0051] The techniques described above allow an operating system designer to specify informational descriptors that devices can implement to provide additional data about themselves—data that is not directly addressed by the USB specification. For example, the techniques described above allow an operating system to specify the extended configuration descriptor 122 of Fig. 1. The operating system uses the extended configuration descriptor to enumerate all USB devices that support non-standard USB DWG class codes and/or subclass codes to determine one or more device 30 drivers to control the devices. The extended configuration descriptor and procedures to use the extended configuration descriptor to enumerate non-standard compatible ids are described in greater detail below in reference to Fig. 4 and Tables 1-2.

### Exemplary Data Structures

35 [0052] Fig. 4 shows elements of an exemplary data structure for the extended configuration descriptor 122, which is an example of the host-specific request descriptor described above. The extended configuration descriptor 122 includes a header section 400 and a function section 402. The header section includes a number of elements 404 that describe the function section. The function section includes one or more control functions 406. Each instance of a 40 control function 406 encapsulates information corresponding to a single function for the peripheral device 114.

[0053] Header section 400 stores information about the remaining portions of the extended configuration descriptor 122. Header section 400 includes the following fields:

- *dwLength*—the total length of the extended configuration descriptor;
- *bcdVersion*—the version number of the extended configuration descriptor;
- *wIndex*—identifies the extended configuration descriptor; and
- *wCount*—the total number of control function sections 406 in the function section 402.

Using this information, the operating system or application program can parse the following function section.

50 [0054] Function section 402 is of variable size because it includes one or more control function sections 406. Each control function section includes information that corresponds to a single function for the peripheral device. The function section includes the following fields:

- *dwSize*—the total length of this control function section;
- *qwCompatibleID*—the compatible ID, or class code for this function;
- *qwSubCompatibleID*—the sub-compatible ID, or subclass code for this function;
- *bTotalInterfaces*—the number of interfaces grouped together to generate this function;
- *blInterfaceNumber*—the interface number;

EP 1 221 653 A2

[0055] The *qwCompatibleID* and *qwSubCompatibleID* values are used to override any USB DWG standard compatible ids and/or sub-compatible ids that may be defined for the device. This means that if USB DWG standard class and/or subclass codes are defined for the USB device, that these standard codes will be ignored in view of the extended configuration descriptor provided compatible and/or sub-compatible ids. The *qwCompatibleID* and *qwSubCompatibleID* values should not be confused with the USB DWG defined class and subclass codes — regardless of whether they are used in the extended configuration descriptor.

[0056] The *bTotalInterfaces* field is a value that represents the total number of interfaces that have been grouped together for this particular function. Each individual interface number (*bInterfaceNumber*) forming the function is represented below it.

Example Extended Configuration Descriptor Use

[0057] For an operating system to use an extended configuration descriptor stored on a device, the device must first provide information to the operating system that indicates that the device supports host-specific device requests. In this implementation, this is accomplished by storing the string descriptor shown in TABLE 1 at index 0xEE on the USB device. The string descriptor defines an operating system descriptor that is returned to the operating system in response to a USB device request as discussed above. In this implementation, the string descriptor index is 0xEE, however it could be stored at some other location in the USB device.

TABLE 1

Example of an Operating System String Descriptor

```
const byte os_descriptor[]={  
    0x12, //length  
    0x03, // string descriptor type  
    'M', 'S', 'F', 'T', '1', '0', '0', //data  
    0x01, //index  
    0x00 //reserved  
};
```

[0058] Next, the extended configuration descriptor is stored in a format specified by the operating system on the firmware of the USB device. The extended configuration descriptor is shown below in TABLE 2 In this implementation, the USB device is a device that is compatible with a Remote Network Device Interface Specification (RNDIS) that is not supported by the USB DWG. (This means that the USB DWG has not supplied any compatible ids for RNDIS devices). In this implementation, the device needs to indicate to the operating system that it is compatible with the RNDIS. To accomplish this, the device implements the following extended configuration descriptor.

45

50

55

TABLE 2

Example of an Extended Configuration Descriptor for a RNDIS  
Compatible USB Device

```

5      const byte mem_stick_descriptor[]={           //header section
10     0x00000044,                                // dwLength (DWORD)
15     0x01000,                                 // bcdVersion
20     0x0004,                                  // wIndex
25     0x0001,                                  // wCount
30
35     // function section
40     0x00000016,                                // wSize
45     "RNDIS",                                 // qwCompatibleID
50     "",                                     // qwSubCompatibleID
55     2,                                      // bTotalInterfaces
60     1,                                      // First Interface in function 1
65     3,                                      // Second interface in function = 3
70   };
75

```

[0059] When the RNDIS compatible device is attached to the USB port on the computer, the operating system retrieves the extended configuration descriptor from the device by using the following API call:

```

30
35   GET_DESCRIPTOR(
40     bmRequestType = 1100 0001B,    // device-to-host data transfer
45     bRequest = 0x01,             // this request
50     wValue = 0x0000,            // string descriptor indication
55     wIndex = 0x0004);          // this index
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995
1000
1005
1010
1015
1020
1025
1030
1035
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100
1105
1110
1115
1120
1125
1130
1135
1140
1145
1150
1155
1160
1165
1170
1175
1180
1185
1190
1195
1200
1205
1210
1215
1220
1225
1230
1235
1240
1245
1250
1255
1260
1265
1270
1275
1280
1285
1290
1295
1300
1305
1310
1315
1320
1325
1330
1335
1340
1345
1350
1355
1360
1365
1370
1375
1380
1385
1390
1395
1400
1405
1410
1415
1420
1425
1430
1435
1440
1445
1450
1455
1460
1465
1470
1475
1480
1485
1490
1495
1500
1505
1510
1515
1520
1525
1530
1535
1540
1545
1550
1555
1560
1565
1570
1575
1580
1585
1590
1595
1600
1605
1610
1615
1620
1625
1630
1635
1640
1645
1650
1655
1660
1665
1670
1675
1680
1685
1690
1695
1700
1705
1710
1715
1720
1725
1730
1735
1740
1745
1750
1755
1760
1765
1770
1775
1780
1785
1790
1795
1800
1805
1810
1815
1820
1825
1830
1835
1840
1845
1850
1855
1860
1865
1870
1875
1880
1885
1890
1895
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030
2035
2040
2045
2050
2055
2060
2065
2070
2075
2080
2085
2090
2095
2100
2105
2110
2115
2120
2125
2130
2135
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
2200
2205
2210
2215
2220
2225
2230
2235
2240
2245
2250
2255
2260
2265
2270
2275
2280
2285
2290
2295
2300
2305
2310
2315
2320
2325
2330
2335
2340
2345
2350
2355
2360
2365
2370
2375
2380
2385
2390
2395
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
2490
2495
2500
2505
2510
2515
2520
2525
2530
2535
2540
2545
2550
2555
2560
2565
2570
2575
2580
2585
2590
2595
2600
2605
2610
2615
2620
2625
2630
2635
2640
2645
2650
2655
2660
2665
2670
2675
2680
2685
2690
2695
2700
2705
2710
2715
2720
2725
2730
2735
2740
2745
2750
2755
2760
2765
2770
2775
2780
2785
2790
2795
2800
2805
2810
2815
2820
2825
2830
2835
2840
2845
2850
2855
2860
2865
2870
2875
2880
2885
2890
2895
2900
2905
2910
2915
2920
2925
2930
2935
2940
2945
2950
2955
2960
2965
2970
2975
2980
2985
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
9999

```

[0060] In response to receiving this request, the USB device returns the extended configuration descriptor to the operating system. Upon receiving the extended configuration descriptor, the operating system determines the non-standard compatible ids based on the extended configuration descriptor. The operating system uses the non-standard compatible ids to load generic (or default) device drivers to control the USB device. This is beneficial because OEMs/IHVs who develop USB devices that are not fully supported by USB DWG standard class and subclass device drivers can specify non-standard class and subclass device drivers to control the USB devices.

[0061] Moreover, unless the DWG has a predefined request that returns one or more multiple interfaces that correspond to a function in a composite USB device, a complex procedure must typically be implemented in traditional systems and procedures to determine the interfaces that correspond to the functions. However, the extended configuration descriptor of this implementation includes each interface to function mapping implemented by a composite device. Thus, in contrast to such traditional systems and procedures, this implementation provides a simple way for a new composite USB device to specify a group of interfaces because the extended configuration descriptor is communicated to a requestor in response to receipt of a single host-specific device request.

[0062] Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

**Claims**

1. A method comprising:

5 providing an extended configuration descriptor in firmware of a USB device, the extended configuration descriptor comprising a set of non-standard class codes ; and responsive to receiving a host-specific device request, communicating the extended configuration descriptor to a requestor.

10 2. A method as recited in claim 1, wherein the set of non-standard class codes includes non-standard subclass codes.

3. A method as recited in claim 1, wherein a non-standard class code comprises a class code or a subclass code that is not defined by the USB DWG.

15 4. A method as recited in claim 1, wherein the extended configuration descriptor further comprises:

a control function section indicating information corresponding to a function for the USB device.

5. A method as recited in claim 1, wherein the extended configuration descriptor further comprises:

20 a header section indicating the number of control function sections for which mappings exist in the extended configuration descriptor; and, one or more control function sections, each control function section indicating information corresponding to a single function for the USB device.

25 6. One or more computer-readable media containing a computer executable program that performs a method as recited in claim 1.

7. A method comprising:

30 querying a USB device using a host-specific device request to obtain a descriptor indicating a set of non-standard class codes; determining one or more compatible device drivers based on the set of codes indicated by the descriptor; and loading the one or more compatible device drivers to control the USB device.

35 8. A method as recited in claim 7, wherein the querying comprises:

40 communicating a standard USB request to the USB device; and returning the extended configuration descriptor in response to the standard USB request, the extended configuration descriptor corresponding in the USB device to a host specific device request.

9. A method as recited in claim 7, wherein the set of non-standard class codes includes non-standard subclass codes.

45 10. A method as recited in claim 7, wherein a non-standard class code comprises a class code or a subclass code that is not defined by the USB DWG.

11. A method as recited in claim 7, wherein the extended configuration descriptor comprises:

50 a control function section indicating information corresponding to a function for the USB device.

12. A method as recited in claim 7, wherein the extended configuration descriptor comprises:

55 a header section indicating the number of control function sections for which mappings exist in the extended configuration descriptor; and, one or more control function sections, each control function section indicating information corresponding to a single function for the USB device.

13. One or more computer-readable media containing a computer executable program that performs a method as

recited in claim 7.

14. In a USB device that responds to device requests from a host, the device requests including USB-specific device requests with corresponding USB-specified request codes and device-specific device requests with corresponding device-specified request codes, the USB-specific device requests including a GET\_DESCRIPTOR device request with a corresponding GET\_DESCRIPTOR request code, a method of implementing a host-specific device request to determine one or more device drivers to control the USB device, the method comprising:
  - receiving a GET\_DESCRIPTOR device request that specifies a predetermined index;
  - responding to the GET\_DESCRIPTOR device request by returning an extended configuration descriptor that corresponds in the USB device to the host-specific device request for a device-specific request code, the extended configuration descriptor specifying a non-standard class code that identifies a device driver to control the USB device.
15. A method as recited in claim 14, wherein the extended configuration descriptor comprises:
  - a control function section indicating information corresponding to a function for the USB device.
16. A method as recited in claim 14, wherein the extended configuration descriptor comprises device specific information, the device specific information comprising:
  - a header section indicating the number of control functions for which mappings exist in the extended configuration descriptor; and,
  - one or more control function sections, each control function section indicating information corresponding to a single function for the USB device.
17. One or more computer-readable media containing a computer executable program that performs a method as recited in claim 14.
18. A computer-readable medium containing computer-executable instructions utilized by an application program to interact with a USB control device, the computer-executable instructions comprising:
  - receiving a request from an application program for a set of non-standard class codes and subclass codes that correspond to the USB control device;
  - querying the USB control device to obtain an extended configuration descriptor, the extended configuration descriptor corresponding to a host-specific device request that identifies the set of non-standard class codes and subclass codes; and
  - returning the obtained extended configuration descriptor to the requesting application program.
19. A computer-readable storage medium as recited in claim 18, further comprising:
  - determining, by the requesting application program, one or more default device drivers based on the returned extended configuration descriptor; and
  - loading the one or more default device drivers to control the control device.
20. A computer-readable storage medium as recited in claim 18, wherein the obtained extended configuration descriptor comprises:
  - a control function section indicating information corresponding to a function for the USB device.
21. A computer-readable storage medium as recited in claim 18, wherein the obtained extended configuration descriptor comprises:
  - a header section indicating the number of control functions for which mappings exist in the extended configuration descriptor; and,
  - one or more control function sections, each control function section indicating information corresponding to a single function for the USB device.

EP 1 221 653 A2

22. A computer comprising one or more computer-readable media as recited in claim 18.
23. One or more computer-readable media containing a computer-executable program for use in conjunction with a USB device that responds to device requests from the program, the device requests including USB-specific device requests with corresponding USB-specified request codes and device-specific device requests with corresponding device-specified request codes, the program comprising:
- 5 sending a request to the USB device for an extended configuration descriptor indicating one or more control functions that correspond to the USB device;
- 10 receiving the extended configuration descriptor from the USB device in response to the request, wherein the extended configuration descriptor corresponds in the USB device to a host-specific device request.
24. One or more computer-readable media as recited in claim 23, wherein the extended configuration descriptor comprises:
- 15 a control function section indicating information corresponding to a function for the USB device.
25. One or more computer-readable media as recited in claim 23, wherein the extended configuration descriptor comprises:
- 20 a header section indicating the number of control functions for which mappings exist in the extended configuration descriptor; and,  
one or more control function sections, each control function section indicating information corresponding to a single function for the USB device.
26. One or more computer-readable media as recited in claim 23, wherein the program further comprises:
- 25 determining one or more compatible device drivers based on the received extended configuration descriptor.
27. A computer comprising one or more computer-readable media as recited in claim 23.
28. A USB device comprising:
- 30 a processor;  
a port coupled to the processor;  
a memory coupled to the processor;  
an extended configuration descriptor stored in the memory, the extended configuration descriptor comprising information that identifies a set of non-standard compatible IDs corresponding to the USB device; and  
40 a control program module stored in the memory, the control program module being configured to send the extended configuration descriptor to a requestor responsive to receiving a host-specific device request at the port.
29. A USB device as recited in claim 28, wherein the set non-standard compatible IDs comprise class codes and/or subclass codes that are not defined by the USB DWG.
30. A USB device as recited in claim 28, wherein the extended configuration descriptor comprises:
- 45 a control function section indicating information corresponding to a function for the USB device.
31. A USB device as recited in claim 28, wherein the extended configuration descriptor comprises:
- 50 a header section indicating the number of control functions for which mappings exist in the extended configuration descriptor; and,  
one or more control function sections, each control function section indicating information corresponding to a single function for the USB device.
32. A USB device as recited in claim 28, wherein the USB device further comprises:

a plurality of logical devices, each logical device (LD) being a sub-device of the USB device;

wherein the extended configuration descriptor further comprises a set of information corresponding to a plurality of interfaces, each LD corresponding to one or more of the interfaces; and

5 wherein the control program module is further configured to communicate the set of information to a requestor responsive to receiving a single request at the port.,

33. A USB device as recited in claim 28, wherein the set of non-standard compatible IDs are not supported by the DWG.

10 34. A USB device as recited in claim 28, wherein the extended configuration descriptor further comprises:

a control function section indicating information corresponding to a function for the USB device.

35. A USB device recited in claim 28, wherein the extended configuration descriptor further comprises:

15 a header section indicating the number of control functions for which mappings exist in the extended configuration descriptor; and,  
one or more control function sections, each control function section indicating information corresponding to a single function for the USB device.

20 36. A computer-readable medium having stored thereon a data structure, comprising:

a first data field comprising data indicating a count indicating the number of USB control functions for which mappings exist in a descriptor; and

25 one or more second data fields, the number of second data fields based on the count, each second data field comprising data corresponding to a single function for a USB device.

37. A computer-readable medium as recited in claim 36, wherein the first data field further comprises:

30 a total descriptor length indication; and  
a descriptor version indication.

38. A computer-readable medium as recited in claim 36, wherein each second data field further comprises a class code to override a standard USB DWG standard class code.

35 39. A computer-readable medium as recited in claim 38, wherein the class code is selected from a group of identifications comprising a compatible class code and a sub-compatible class code.

40. A computer-readable medium as recited in claim 36, wherein each second data field further comprises:

45 a control function length indication;  
a total number of interfaces indication, the total number of interfaces being grouped together to generate a control function; and  
an interface number.

46

50

55

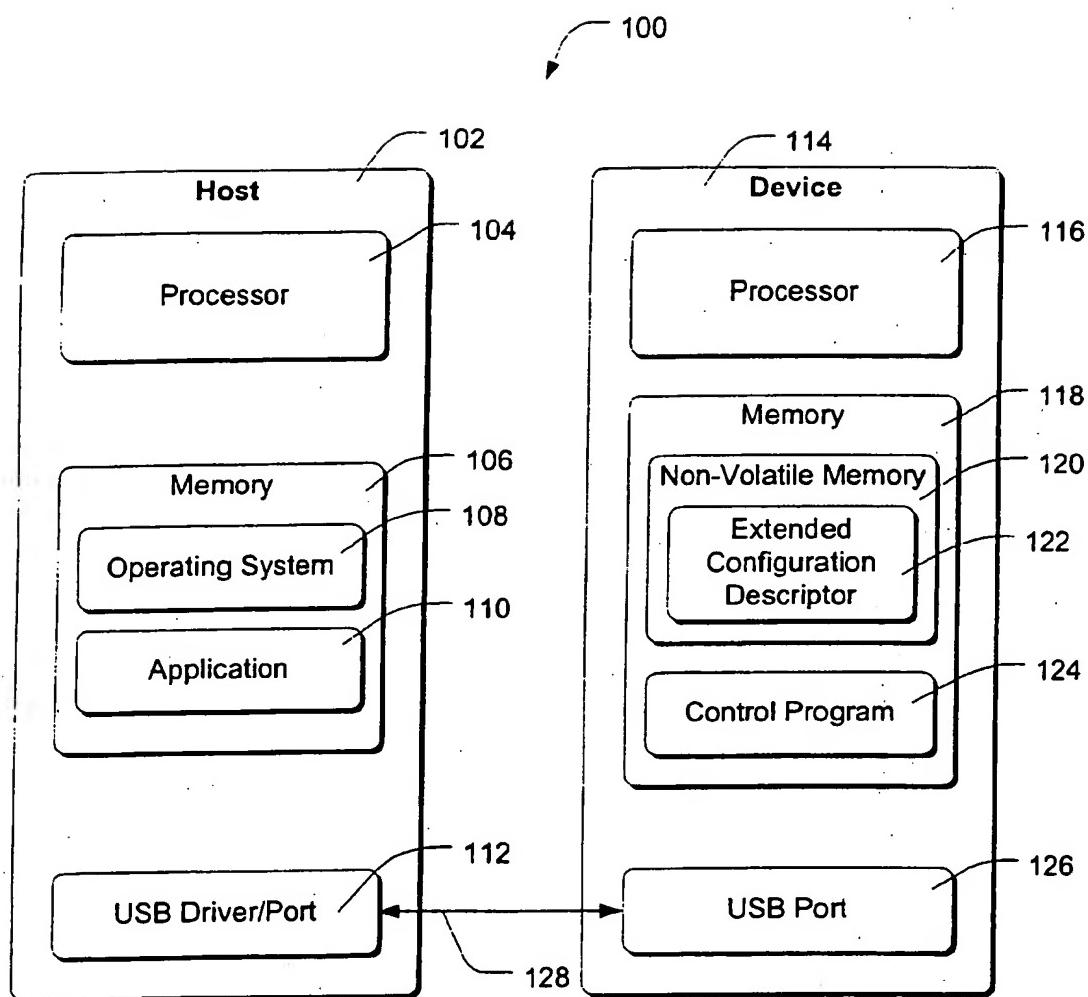


Fig. 1

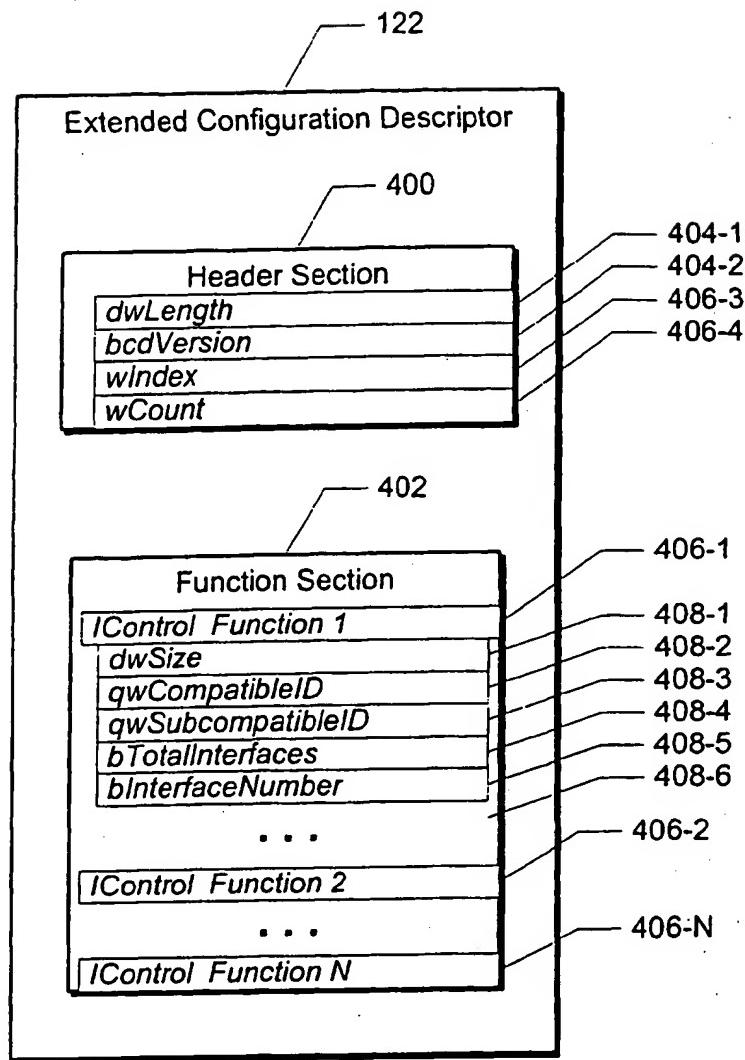
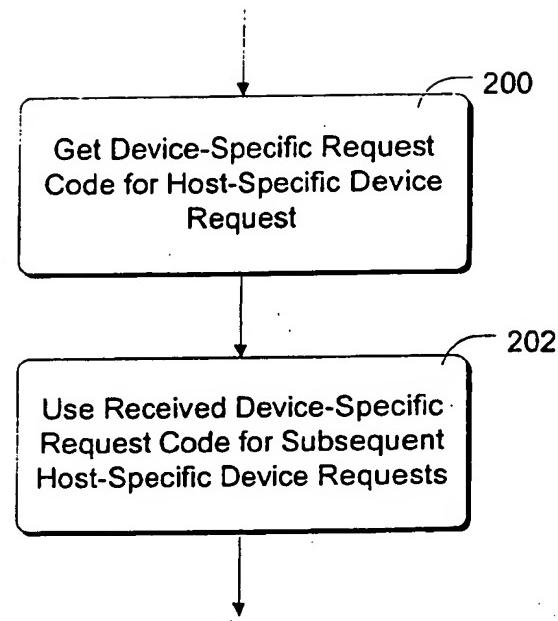
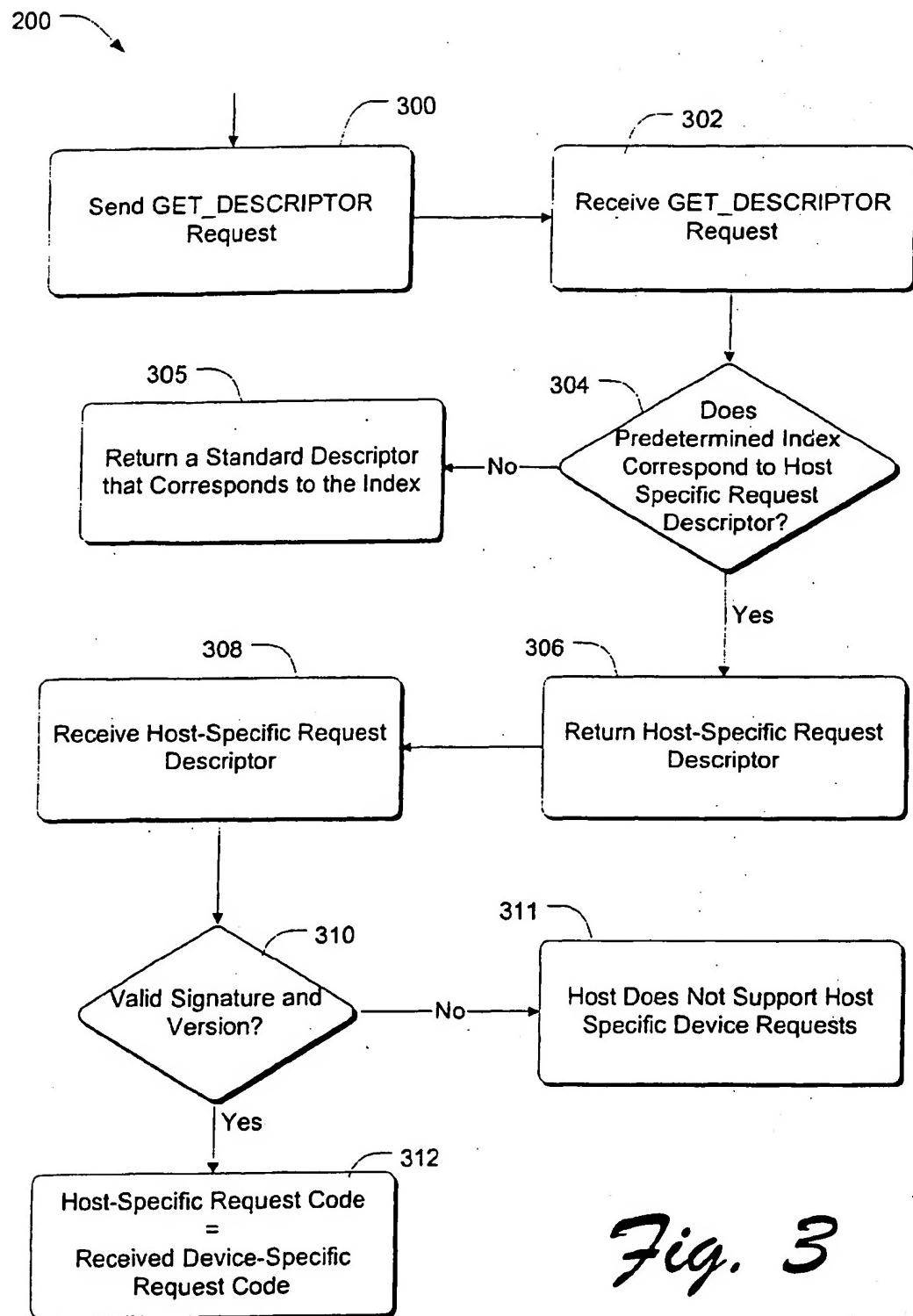


Fig. 4



*Fig. 2*



*Fig. 3*